Discrete Event Control with CFC

Primer A. Autor, Segundo B. Autor*, y Tercer C. Autor***, Miembro,*IEEE*

**El Departamento, La Institución*, primero@correo.com, segundo@correo.es
***Otro Departamento, Otra Institución*, tercero@correo.net
(para el proceso de revisión por favor omitir los nombres de los autores)

Abstract **In this paper it is proposed a discrete event control system representation methodology to design, implement and operate sequential systems, as an alternative to the sequential function chart (SFC) description method. Such method uses the standard language ST (structured text) of the IEC 61131-3 associated to the standard language developed by the Foundation Fieldbus, by associating ST based descriptions language to some function blocks on the basis of a continuous function charts (CFC) or function block diagrams (FBD).**

***Index Terms*—Continuous function chart, discrete event systems, Foundation Fieldbus, IEC-1131-3, sequential function chart, structured text.**

I. Introductión

DISCRETE event control systems has been designed using mainly Petri Nets [1,5] and standardised tools, such as Grafcet [2] and SFC as formal methods [3]. With the use of tools to design and implement continuous control such as CFC [7], there exists a real possibility for implementing discrete event control systems by associating logic function blocks programmed under ST language to a control system description based in CFC.

Proposed idea does not emerge as a solution to substitute SFC based control systems or Grafcet-based batch controlled processes but to serve as an alternative to embed short modules based in sequential control into a general CFC based control system described by means of FBD.

This method is intended to serve in avoiding the combination of conventional SFC description with CFC based control systems in special cases such as the task of diagnosing a control module [6]. Following sections deals respectively with the necessary function blocks characteristics, a brief description of a SFC methodology description, and finally it is presented the method to implement discrete event control under FBD's description t method.

II. Function Blocks to Implement a CFC Based Discrete Event Control System

The minimum necessary function block classes to implement a CFC based sequential discrete event control system are [7]:

- Calculation/Logic (CALC) function block
- Set/Reset Flip-flop (SR) function block
- Action (ACT) function block

A brief description of mentioned function blocks is presented bellow

Calculation/Logic Function Block

The Calculation/Logic (CALC) function block allows you to specify an expression that determines the block's output. Mathematical functions, logical operators, constants, parameter references, and I/O reference values can be used in the expression. Figure 1 shows the symbolic Calculation/Logic function block representation.

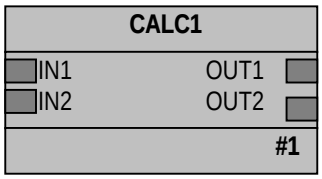

Fig. .1.Calc/Logic function block

IN1 through IN[n] are the inputs to the block (as many as 16 inputs).
OUT1 through OUT[n] are the block outputs (as many as 16 outputs).
Figure 2 shows the schematic diagram of Calculation/Logic function block.

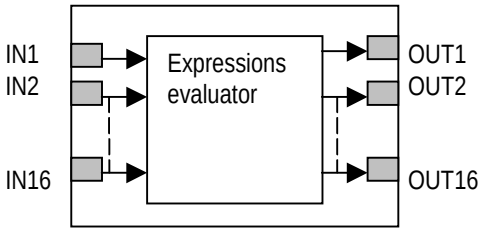

Fig. 2. Schematic diagram of Calculation/Logic function block

Set/Reset Flip-flop Function Block

The Set/Reset Flip-flop (SR) function block generates a discrete output value based on NAND logic of set and reset inputs:
When the reset input is False (0) and the set input is True (1), the output is True. The output remains True until the reset input is True and the set input is False.

When the reset input is True, the output is equal to the set input. When both inputs are True, the output is True. When both inputs become False, the output remains at its last state and can be either True or False. Figure 3 shows the symbolic representation of Set/Reset function block.

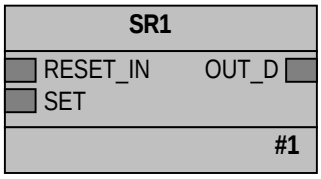

Fig. 3. Set/Reset function block

Set/Reset Flip-flop Function Block has the following input/output connection pins:
RESET_IN is the reset discrete input value and status.
SET is the set discrete input value and status.
OUT_D is the discrete output signal and status.

Action function block

The Action (ACT) function block evaluates an expression when the input value is True. Mathematical functions, logical operators, and constants can be used in the expression. Figure 4 shows the symbolic representation of the Action Function Block.

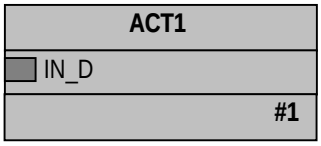

Fig. 4. Action Function Block

IN_D is the discrete input value and status that initiates expression evaluation.

III. SFC

SFCs are types of module algorithms that are useful for controlling time-event sequences, such as startup or shutdown of a process [4,6]. SFCs are made up of steps and transitions. Steps contain a set of actions. A transition allows a sequence to proceed from one step to the next when the transition condition is true.

Each time the SFC scans, the system evaluates the active steps and transitions. When a transition evaluates as True, the step prior to the transition is made inactive and the step following the transition becomes active.

As a general rule, in defining a SFC, it can be found helpful to first define the steps in the process, and then identify the conditions that must be met before proceeding from step to step. As have been stated, SFCs are useful for representing and controlling sequential processing behaviour. They are best at controlling strategies that require multiple states

A sequence in an SFC is drawn as a series of steps and transitions. Steps are represented by boxes and transitions by vertical lines with crosses attached. Each step contains a set of actions that affect the process. At any given time, one or more of the steps and transitions can be active. Each time the SFC scans, the active steps and transitions are evaluated. When a transition evaluates as TRUE (for example, the transition condition is met), the steps prior to the transition are made inactive and the step(s) following the transition become active.

Transitions allow single-stream or parallel execution of logic within the SFC. In an SFC, it can be used divergent paths to enter alternative sequences by using a sequence select divergence, which looks similar to the following shown at figure 5

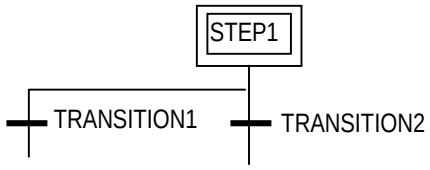

Fig. 5. Example of a Sequence Select Divergence

To converge paths again, it can be used a sequence select convergence, which looks similar to the following at figure 6.

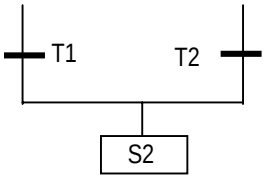

Fig. 6. Example of a Sequence Select Convergence

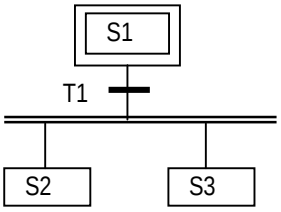

Fig. 7 Example of a Parallel Divergence

It can also be executed simultaneous, or parallel, sequences by using a parallel divergence, which looks similar to the one in figure 7. A parallel convergence brings the parallel sequences back together. The flow is then controlled from one step to the next using transitions, where transitions identify conditions

that must be met before a sequence can proceed to the next step.

IV. USING CFC METHOD TO IMPLEMENT SFC APPLICATIONS

Scheduling a SFC application by means of the proposed CFC representation requires the set of function blocks mentioned in section II.

The Calc/Logic function block, have the ability to enable and disable transitions, to evaluate the logic state of any transition, and to enable and disable steps. Such abilities are implemented under the ST language.

Set/Reset function block, activate and deactivate action steps when a enabled or disabled transition reach the true or false logic state.

The Action function block describe the action associated to any step, which can be a physical action and/or any computation procedure. The actions into steps can be realised then by means of Calc/Logic function blocs and Action function blocks simultaneously.

In order to show the procedure we proceed via an example. Let us consider a SFC application such as the one shown in figure 8.

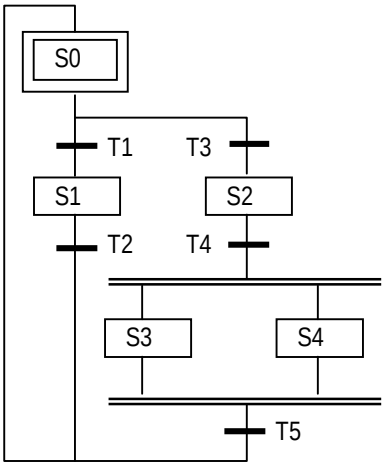

Fig. 8. SFC based description of a discrete event system.

A transition allows a sequence to proceed from one step to the next when the transition condition is true, if and only if the transition is enabled. The transition is enabled by the precedent active step. So that, this assertment can be described under ST language as:

```
IF S(i-1) AND T(i) THEN
  S(i):=TRUE;
  S(i-1):=FALSE;
END_IF                                    (1)
```

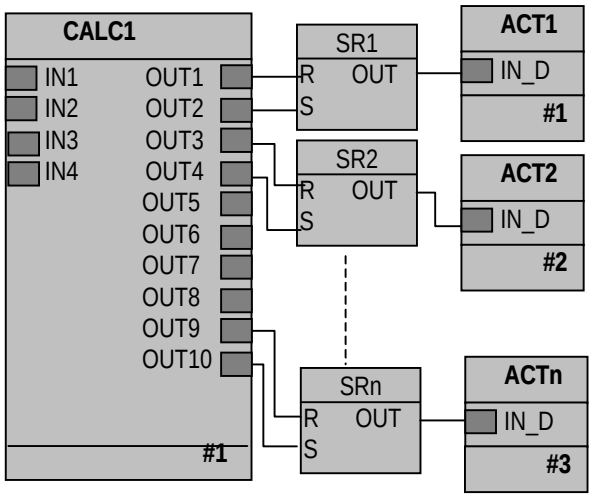

Fig. 9. CFC based implementation of the SCF example.

Calc/Logic function block allows us to process past instructions described by (1). The outputs of the Calc/Logic function block OUT(i) are activated and deactivated according expressions (1) to set and reset the corresponding steps, as shown in figure 9. When a step is set, then the Action function bloc associated operates accordingly.

In figure 9 only Action function blocks are associated to step activities but in case of intensive computation procedures, Cal/Logic function blocks could be applied.

The following text in ST language implements the SFC based description shown in figure 8. Such text is supported by the Calc/Logic function bloc shown in figure 9.

```
IF S(0) AND T(1) THEN
  S(1):=TRUE;
  S(0):=FALSE;
END_IF;

IF S(0) AND T(3) THEN
  S(2):=TRUE;
  S(0):=FALSE;
END_IF;

IF S(1) AND T(2) THEN
  S(0):=TRUE;
  S(1):=FALSE;
END_IF;

IF S(2) AND T(4) THEN
  S(3):=TRUE;
  S(4):=TRUE;
  S(2):=FALSE;
END_IF;

IF (S(3) OR S(4)) AND T(5) THEN
  S(0):=TRUE;
  S(3):=FALSE;
  S(4):=FALSE;
END_IF;
```

Activated steps by the Calc/Logic function bloc outputs, set or reset the Set/Reset function blocks, which send a logic signal to each Action function block. The action function bloc is allowed to operate when its input is true. Alternatively or simultaneously, Calc/Logic function blocks, used as procedure computation and action blocks, will be activated by its enabled inputs as shown at figure 10.

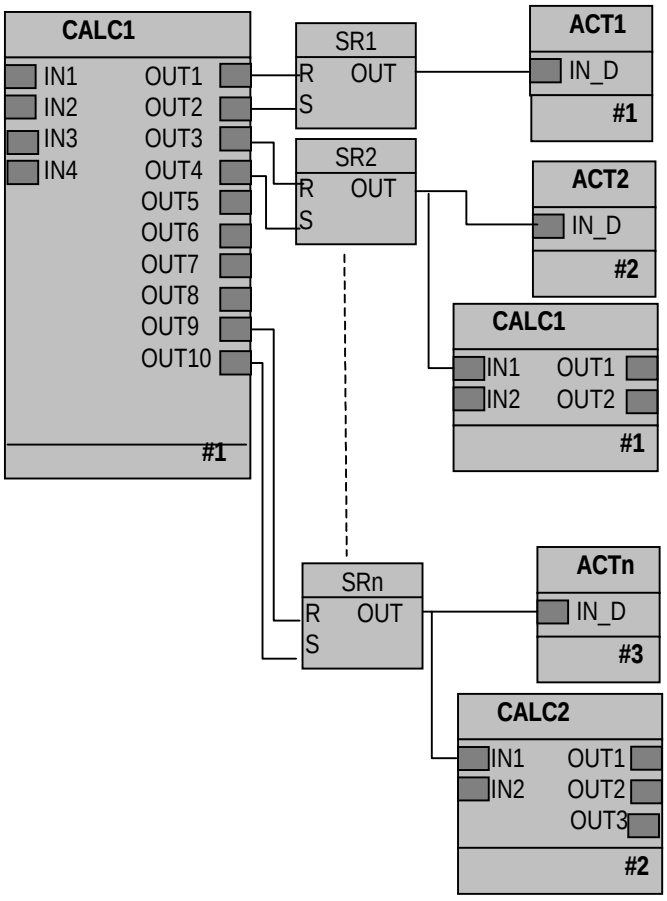

Fig. 10. CFC based implementation of the SCF example with Calc/Logic function blocks as action blocks.

Actions defined in figure 10 by means of Calc/Logic function blocks are subjected to the state of precedent block Sri, depending whether it is enabled or disable. So that, a enabled SRi function block enables a Calc/Logic function block to operate according action instructions defined into a IF THEN instruction which looks like following source listing fro function block CALC2 of figure 10:

```
IF IN1 THEN
(* Compute and apply actions in actual
step. For example :*);
  OUT1:=TRUE;
  OUT2:=Pi*TAN(IN2);
  OUT3:=sqrt(IN2);
  I:=0;
  (*WHILE Loop to generate a ramp *)
  WHILE I<50 DO;
    OUT3:=OUT3+0.25*I;
    I:= I+1;
    END_WHILE;
END_IF;
```

V. CONCLUSIONS

System integrators appreciate the existence of powerful tools to implement complex discrete event and hybrid control systems. But at same time, they are interested in control developing tools easy to understand and operate.

The described alternative is not an attractive solution to solve efficiently large SFC based batch control processes. This is rather useful in solving discrete event processes associated to control modules where short hybrid control tasks are needed.

Doesn't require special or complex knowledge and the clarity of the method contribute to make such method applicable.

REFERENCIAS

[1] R. David. H. Alla, "Petri Nets and Grafcet: Tools for Modelling Discrete Event Systems", Prentice Hall, London, 1992.

[2] K. E. Arcen, "Sequential Function Charts for Knowledge-based Real Time Applications", Proc. of the 3d IFAC Workshop on AI in Real Time Control, California, 1991.

[3] IEC 61131-3, "Programmable Controllers-Part 3, Their Programming Languages", International Electro-technical Commission, Publication 61131-3, 1993

[4] L. Marce, P. Le Parc, "Defining the Semantics of languages for Programmable Controllers with Synchronous Processes, Control Engineering Practice, Vol 1 (1993) pp. 79-84

[5] A.H. Jones (et al.), "A General Methodology for Converting Petri Nets into Ladder Logic: The TPL Methodology", Journal of Intelligent Manufacturing 5 (1996), pp. 103-120.

[6] "SFC++:a tool for developing distributed real-time control software", Ed. Elsevier, Microprocessors and Mycrosystems, 23 (1999), pp. 75-84

[7] Fisher-Rosemount Systems, Inc. "DeltaV Books on-line", Copyright © 1994-2001, Emerson Process Management.